

This application note illustrates the different functions of the Programmable Counter Array (PCA) which are available on the 83C51FA and 83C51FB. Included are cookbook samples of code in typical applications to simplify the use of the PCA. Since all the examples are written in assembly language, it is assumed the reader is familiar with ASM51. For further information on these products or ASM51, refer to the Embedded Controller Handbook (Vol. 1).

PCA OVERVIEW

The major new feature on the 83C51FA and 83C51FB is the Programmable Counter Array. The PCA provides monitoring capabilities with less CPU intervention than the standard timer/counters. Its advantages include reduced software overhead and improved accuracy.

The PCA consists of a dedicated timer/counter which serves as the time base for an array of five compare/capture modules. Figure 1 shows a block diagram of the PCA. Notice that the PCA timer and modules are all 16-bits. If an external event is associated with a module, that function is shared with the corresponding Port 1 pin. If the module is not using the port pin, the pin can still be used for standard I/O.

83C51FA/FB PCA Cookbook

BETSY JONES
ECO APPLICATIONS ENGINEER

July 1988

Each of the five modules can be programmed in any one of the following modes:

- Rising and/or Falling Edge Capture
- Software Timer
- High Speed Output
- Watchdog Timer (Module 4 only)
- Pulse Width Modulator

All of these modes will be discussed later in detail. However, let's first look at how to set up the PCA timer and modules.

PCA TIMER/COUNTER

The timer/counter for the PCA is a free-running 16-bit timer consisting of registers CH and CL (the high and low bytes of the count values). It is the only timer which can service the PCA. The clock input can be selected from the following four modes:

- oscillator frequency $\div 12$ (Mode 0)
- oscillator frequency $\div 4$ (Mode 1)
- Timer 0 overflows (Mode 2)
- external input on P1.2 (Mode 3)

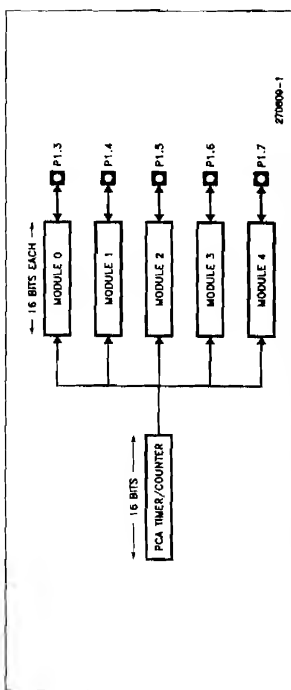


Figure 1. PCA Timer/Counter and Compare/Capture Modules

The table below summarizes the various clock inputs for each mode at two common frequencies. In Mode 0, the clock input is simply a machine cycle count, whereas in Mode 1 the input is clocked three times faster. In Mode 2, Timer 0 overflows are counted allowing for a range of slower inputs to the timer. And finally, if the input is external the P_{CA} timer counts 1-to-0 transitions with the maximum clock frequency equal to 1/4 x oscillator frequency.

Table 1. PCA Timer/Counter Inputs

PCA Timer/Counter Mode	Clock Increments	
	12 MHz	15 MHz
Mode 0: fosc / 12	1 μ sec	0.75 μ sec
Mode 1: fosc / 4	330 nsec	250 nsec
Mode 2*: Timer 0 Overflows		
8-bit mode	256 μ sec	192 μ sec
16-bit mode	65 msec	49 msec
8-bit auto-reload	1 to 255 μ sec	0.75 to 191 μ sec
Mode 3: External Input MAX	0.66 μ sec	0.50 μ sec

*In Mode 2, the overflow interrupt for Timer 0 does not need to be enabled.

Special Function Register CMOD contains the Count, Pulse Select bits (CPS1 and CPS0) to specify the PCA timer input. This register also contains the ECF bit which enables an interrupt when the counter overflows. In addition, the user has the option of turning off the PCA timer during Idle Mode by setting the Counter Idle bit (CIDL). This can further reduce power consumption by an additional 30%.

CMOD: Counter Mode Register

CIDL	WDTE	—	—	—	CPS1	CPS0	ECF
------	------	---	---	---	------	------	-----

Address = 0D9H

Not Bit Addressable

NOTE:

The user should write 0s to unimplemented bits. These bits may be used in future MCS-51 products to invoke new features, and in that case the inactive value of the new bit will be 0. When read, these bits must be treated as don't-cares.

Reset Value = 00XX X000B

Table 2 lists the values for CMOD in the four possible timer modes with and without the overflow interrupt enabled. This list assumes that the PCA will be left running during Idle Mode.

Table 2. CMOD Values

PCA Count Pulse Selected	CMOD value	
	without interrupt enabled	with interrupt enabled
Internal clock, Fosc/12	00 H	01 H
Internal clock, Fosc/4	02 H	03 H
Timer 0 overflow	04H	05 H
External clock at P1.2	06 H	07 H

The CCON register shown below contains the Counter Run bit (CR) which turns the timer on or off. When the PCA timer overflows, the Counter Overflow bit (CF) gets set. CCON also contains the five event flags for the PCA modules. The purpose of these flags will be discussed in the next section.

CCON: Counter Control Register

CF	CR	—	CCF4	CCF3	CCF2	CCF1	CCF0
Address = 0DBH							
Bit Addressable							
Reset Value = 00X0 0000B							

The PCA timer registers (CH and CL) can be read and written to at any time. However, to read the full 16-bit timer value simultaneously requires using one of the PCA modules in the capture mode and logging a port pin in software. More information on reading the PCA timer is provided in the section on the Capture Mode.

COMPARE/CAPTURE MODULES

Each of the five compare/capture modules has a mode register called CCAPMn (n = 0,1,2,3,or 4) to select which function it will perform. Note the EOCFn bit which enables an interrupt to occur when a module's event flag is set.

CCAPMn: Compare/Capture Mode Register

—	ECOMn	CAPPn	CAPn	MATn	TOGn	PWMn	EOCFn
Address = 0DAH (n = 0)							
0DBH (n = 1)							
0DCH (n = 2)							
0DDH (n = 3)							
0DEH (n = 4)							
Reset Value = X000 0000B							

Table 3 lists the CCAPMn values for each different mode with and without the PCA interrupt enabled; that is, the interrupt is optional for all modes. However, some of the PCA modes require software servicing. For example, the Capture mode is optional for all modes. In the Capture mode, the user must write to the CCAPMn register to enable the purpose of the Software Timer mode is to generate interrupts in software so it would be useless not to have the interrupt enabled. The PWM mode, on the other hand, does not require CPU intervention so the interrupt is normally not enabled.

Table 3. Compare/Capture Mode Values

Module Function	CCAPMn Value	
	without interrupt enabled	with interrupt enabled
Capture Positive only	20H	21 H
Capture Negative only	10H	11 H
Capture Pos. or Neg.	30H	31 H
Software Timer	48H	49 H
High Speed Output	4CH	4D H
Watchdog Timer	48 or 4CH	—
Pulse Width Modulator	42 H	43H

It should be mentioned that a particular module can change modes within the program. For example, a module might be used to sample incoming data. Initially it could be set up to capture a falling edge transition. Then the same module can be reconfigured as a software timer to interrupt the CPU at regular intervals and sample the pin.

Each module also has a pair of 8-bit compare/capture registers (CCAPnH, CCAPnL) associated with it. These registers are used to store the time when a capture event occurred or when a compare event should occur. Register values are limited to 255, based on the free-running PCA timer (CH and CL). For the PWM mode, the high byte register CCAPnH controls the duty cycle of the waveform.

When an event occurs, a flag in CCON is set for the appropriate module. This register is bit addressable so that event flags can be checked individually.

CCON: Counter Control Register									
CF	GR	—	CCF4	CCF3	CCF2	CCF1	CCF0	Reset Value = 0000 0000B	
Address = 0DBH								Bit Addressable	

These five event flags plus the PCA timer overflow flag share an interrupt vector as shown below. These flags are not cleared when the hardware vectors to the PCA interrupt address (0031H) so that the user can determine which event caused the interrupt. This also allows the user to define the priority of servicing each module.

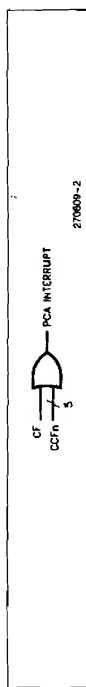


Figure 2. PCA Interrupt

An additional bit was added to the Interrupt Enable (IE) register for the PCA interrupt. Similarly, a high priority bit was added to the Interrupt Priority (IP) register.

IE: Interrupt Enable Register									
EA	EC	ET2	ES	ET1	EX1	ET0	EX0	Reset Value = 0000 0000B	
Address = 0A8H								Bit Addressable	
IP: Interrupt Priority Register									
—	PPC	PT2	PT1	PT0	PX1	PX0	PT0	Reset Value = X000 0000B	
Address = 0B8H								Bit Addressable	

Remember, each of the six possible sources for the PCA interrupt must be individually enabled as well—in the CCAPnM register for the modules and in the CCON register for the timer.

CAPTURE MODE

Both positive and negative transitions can trigger a capture with the PCA. This allows the PCA flexibility to measure periods, pulse widths, duty cycles, and phase differences on up to five separate inputs. This section gives examples of all these different applications.

Figure 3 shows how the PCA handles a capture event. Using Module 0 for this example, the signal is input to P1.3. When a transition is detected on that pin, the 16-bit value of the PCA timer (CCAP0H, CCAP0L) is loaded into the capture registers (CCAP0H, CCAP0L). Module 0's event flag is set and an interrupt is flagged. The interrupt will then be generated if it has been properly enabled.

In the interrupt service routine, the 16-bit capture value must be saved in RAM before the next event capture occurs; a subsequent capture will write over the first capture value. Also, since the hardware does not clear the event flag, it must be cleared in software.

The time it takes to service this interrupt routine determines the resolution of back-to-back events with the same PCA module. To store two 8-bit registers and clear the event flag takes at least 9 machine cycles. That includes the call to the interrupt routine. At 12 MHz, this routine would take less than 10 microseconds. However, depending on the frequency and interrupt latency, the resolution will vary with each application.

Measuring Pulse Widths

To measure the pulse width of a signal, the PCA module must capture both rising and falling edges (see Figure 4). The module can be set up to capture a rising edge if it is known which edge will occur first. However, if this is not known, the user can select which edge will trigger the first capture by choosing the proper mode for the module.

Listing 1 shows an example of measuring pulse widths. (It's assumed the incoming signal matches the one in Figure 4.) In the interrupt routine the first set of capture values are stored in RAM. After the second capture, a subtraction routine calculates the pulse width in units of PCA timer ticks. Note that the subtraction does not have to be completed in the interrupt service routine. Also, this example assumes that the two capture events will occur within 216 counts of the PCA timer, i.e. rollovers of the PCA timer are not counted.

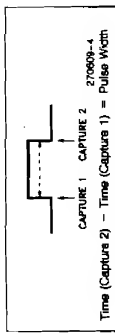


Figure 4. Measuring Pulse Width

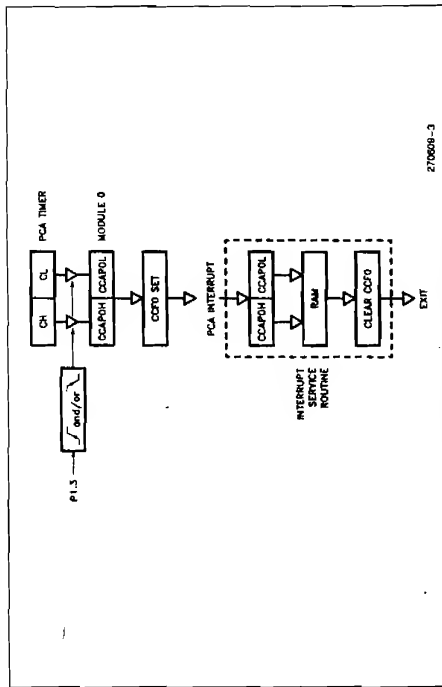


Figure 3. PCA Capture Mode (Module 0)

Listing 1. Measuring Pulse Widths

```

; RAM locations to store capture values
CAPTURE DATA 30H
PULSE_WIDTH DATA 32H
FLAG BIT 20H.0

;
ORG 0000H
JMP PCA_INIT
ORG 0003H
JMP PCA_INTERRUPT
PCA_INIT:
    MOV CMO, #00H
    MOV CH, #00H
    MOV CL, #00H
    ; Initialize PCA timer
    ; Input to timer = 1/12 X Fosc

;
; Initialize Module 0 in capture mode
; Capture positive edge first
; for measuring pulse width
MOV CCAPMO, #21H

;
; Enable PCA interrupt
SETB EC
; Turn PCA timer on
SETB RA
; clear test flag
CLR FLAG
;
; Main program goes here
;
; This example assumes Module 0 is the only PCA module
; being used. If other modules are used, software must
; check which module's event caused the interrupt.

PCA_INTERRUPT:
    CLR CFO
    ; Clear Module 0's event flag
    JB FLAG, SECOND_CAPTURE
    ; Check if this is the first
    ; capture or second

    FIRST_CAPTURE:
    MOV CAPTURE, CCAPOL
    MOV CAPTURE+1, CCAPOH
    MOV CCAPMO, #11H
    ; Save 16-bit capture value
    ; in RAM
    ; Change module to now capture
    ; falling edges
    SETB FLAG
    ; Signify 1st capture complete
    RETI

;
SECOND_CAPTURE:
    PUSH ACC
    PUSH PSW
    CLR C
    MOV A, CCAPOL
    SUBB A, CAPTURE
    MOV PULSE_WIDTH, A
    MOV A, CCAPOH
    SUBB A, CAPTURE+1
    MOV PULSE_WIDTH+1, A
    ;
    MOV CCAPMO, #21H
    CLR FLAG
    POP PSW
    POP ACC
    RETI

```

Measuring Periods

Measuring the period of a signal with the PCA is similar to measuring the pulse width. The only difference will be the trigger source for the capture mode. In Figure 5, rising edges are captured to calculate the period. The code is identical to Listing 1 except that the capture mode should not be changed in the interrupt routine. The result of the subtraction will be the period.

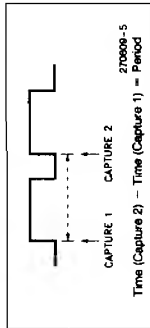


Figure 5. Measuring Period

Measuring Frequencies

Measuring a frequency with the PCA capture mode involves calculating a sample time for a known number of samples. In Figure 6, the time between the first capture and the "Nth" capture equals the sample time T. Listing 2 shows the code for N = 10 samples. It's assumed that the sample time is less than 216 counts of the PCA timer.

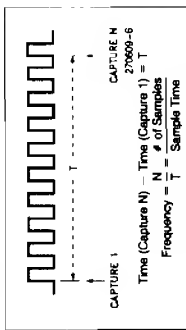


Figure 6. Measuring Frequency

Listing 2. Measuring Frequencies

```

; RAM locations to store capture values
CAPTURE DATA 30H
PERIOD DATA 32H
SAMPLE_COUNT DATA 34H
FLAG BIT 20H.0

;
ORG 0000H
JMP PCA_INIT
ORG 0003H
JMP PCA_INTERRUPT

PCA_INIT:
; Initialization of PCA timer. Module 0, and interrupt is the
; same as in Listing 1. Also need to initialize the sample
; count.
MOV SAMPLE_COUNT, #10D ; N = 10 for this example
;
;
; Main program goes here
;
; This code assumes only Module 0 is being used.
PCA_INTERRUPT:
CLR CCF0 ; Clear module 0's event flag
JB FLAG, NEXT_CAPTURE
;
FIRST_CAPTURE:
MOV CAPTURE, CCAP0L
MOV CAPTURE+1, CCAP0H
SETB FLAG
RETI

;
NEXT_CAPTURE:
DJNZ SAMPLE_COUNT, EXIT
PUSH ACC
PUSH PSW
CLR C
; 16-bit subtraction
SUBB A, CCAP0L
MOV PERIOD, A
MOV A, CCAP0H
SUBB A, CAPTURE+1
MOV PERIOD+1, A
; Reload for next period
MOV SAMPLE_COUNT, #10D
CLR FLAG
POP PSW
POP ACC
EXIT:
RETI

```

The user may instead want to measure frequency by counting pulses for a known sample time. In this case, one module is programmed in the capture mode to count edges (either rising or falling), and a second module is programmed as a software timer to mark the sample time. An example of a software timer is given later. For information on resolution in measuring frequencies, refer to Article Reprint AR-517, "Using the 8051 Microcontroller with Resonant Transducers," in the Embedded Controller Handbook.

Measuring Duty Cycles

To measure the duty cycle of an incoming signal, both rising and falling edges need to be captured. Then the duty cycle must be calculated based on three capture values as seen in Figure 7. The same initialization routine is used from the previous example. Only the PCA interrupt service routine is given in Listing 3.

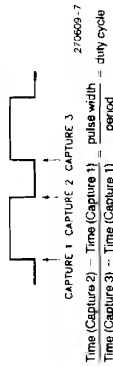


Figure 7. Measuring Duty Cycle

Listing 3. Measuring Duty Cycle

```

; RAM locations to store capture values
CAPTURE DATA 30H
PULSE_WIDTH DATA 32H
PERIOD DATA 34H
FLAG_1 BIT 20H.0
FLAG_2 BIT 20H.1

;
ORG 0000H
JMP PCA_INIT
ORG 0003H
JMP PCA_INTERRUPT
;
PCA_INIT:
; Initialization for PCA timer, module, and interrupt the same
; as in Listing 1. Capture positive edge first, then either
; edge.
;
; Main program goes here
;
; This code assumes only Module 0 is being used.
PCA_INTERRUPT:
CLR CCF0 ; Clear Module 0's event flag
JB FLAG_1, SECOND_CAPTURE
;
FIRST_CAPTURE:
MOV CAPTURE, CCAP0L
MOV CAPTURE+1, CCAP0H
SETB FLAG_1
MOV CCAPW0, #31H ; Signify first capture complete
RETI ; Capture either edge now

```

Listing 3. Measuring Duty Cycle (Continued)

```

; SECOND_CAPTURE:
PUSH PSW
JNB FLAG_2, THIRD_CAPTURE
; Calculate pulse width
; 16-bit subtract
CLR C
MOV A, CCAPOL
SUBB A, CAPTURE
MOV PULSE_WIDTH, A
MOV A, CCAPOH
SUBB A, CAPTURE+1
SETI PULSE_WIDTH+1, A
POP PSW
POP PSW
POP ACC
RETI

; THIRD_CAPTURE:
CLR C
MOV A, CCAPOL
SUBB A, CAPTURE
MOV PERIOD, A
MOV A, CCAPOH
SUBB A, CAPTURE+1
MOV PERIOD+1, A
MOV CCAPMO, #21H
CLR FLAG_1
CLR FLAG_2
POP PSW
POP ACC
RETI

```

After the third capture, a 16-bit by 16-bit divide routine needs to be executed. This routine is located in Appendix B. Due to its length, it's up to the user whether the divide routine should be completed in the interrupt routine or be called as a subroutine from the main program. not assume one signal is leading or lagging the other.

Measuring Phase Differences

Because the PCA modules share the same time base, the PCA is useful for measuring the phase difference

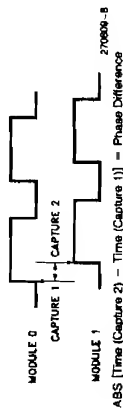


Figure 8. Measuring Phase Differences

Listing 4. Measuring Phase Differences

```

; RAM locations to store capture values
CAPTURE_0 DATA 30H
CAPTURE_1 DATA 32H
PHASE DATA 34H
FLAG_0 BIT 20H.0
FLAG_1 BIT 20H.1

; ORG 0000H
JMP PCA_INIT
ORG 0033H
JMP PCA_INTERRUPT

PCA_INIT:
; Same initialization for PCA timer, and interrupt as
; in Listing 1. Initialize two PCA modules as follows:
;
MOV CCAPMO, #21H ; Module 0 capture rising edges
MOV CCAPML, #21H ; Module 1 same
;
; Main program goes here
;
; This code assumes only Modules 0 and 1 are being used.
PCA_INTERRUPT:
JB CCFO, MODULE_0 ; Determine which module's
JB CCFL, MODULE_1 ; event caused the interrupt
;
MODULE_0:
CLR CCFO ; Clear Module 0's event flag
MOV CAPTURE_0, CCAPOL ; Save 16-bit capture value
MOV CAPTURE_0+1, CCAPOH
JB FLAG_1, CALCULATE_PHASE ; If capture complete on
; Module 1, go to calculation
SETB FLAG_0 ; Signify capture on Module 0
RETI

```


Listing 5. Software Timer

```

: Generate an interrupt in software every 20 msec
:
: Frequency = 12 MHz
: PCA clock input = 1/12 x Fosc -> 1 µsec
:
: Calculate reload value for compare registers:
: 20 msec
: ----- = 20,000 counts
: 1 µsec/count
:
ORG 0000H
JMP PCA_INIT
ORG 0033H
JMP PCA_INTERRUPT
:
PCA_INIT:
: Initialize PCA timer same as in Listing 1
: MOV CCAPM0, #49H : Module 0 in Software Timer mode
: MOV CCAPOL, #LOW(20000) : Write to low byte first
: MOV CCAPOH, #HIGH(20000)
:
: SETB EC : Enable PCA interrupt
: SETB EA
: SETB CR : Turn on PCA timer
:
: .....
: ..... Main program goes here
: .....
:
PCA_INTERRUPT:
CLR CCF0
PUSH ACC
PUSH PSW
CLR EA
MOV A, #LOW(20000)
ADD A, CCAPOL
MOV CCAPOL, A
MOV A, #HIGH(20000)
ADD A, CCAPOH
MOV CCAPOH, A
SETB EA
:
: Continue with routine
:
POP PSW
POP ACC
RETI

```

HIGH SPEED OUTPUT

The High Speed Output (HSO) mode toggles a port pin when a match occurs between the PCA timer and the pre-loaded value in the compare registers (see Figure 10). The HSO mode is more accurate than toggling pins in software because the toggle occurs *before* branching to an interrupt, i.e. interrupt latency will not effect the accuracy of the output. In fact, the interrupt is optional. Only if the user wants to change the time for the next toggle is it necessary to update the compare registers. Otherwise, the next toggle will occur when the PCA timer rolls over and matches the last compare value. Examples of both are shown.

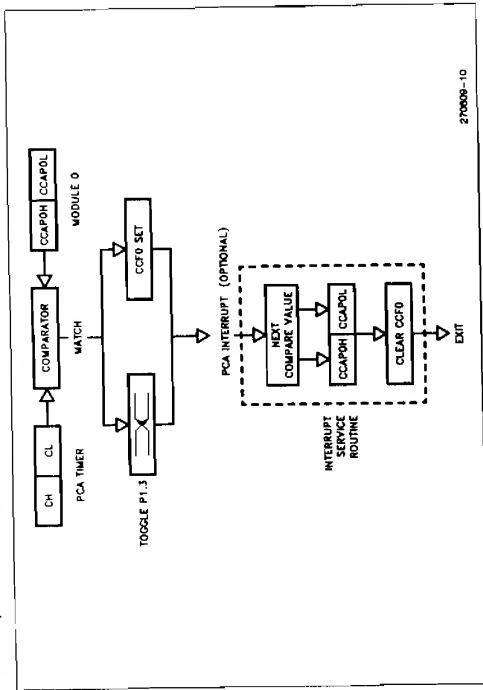


Figure 10. High Speed Output Mode (Module 0)

Without any CPU intervention, the fastest waveform the PCA can generate with the HSO mode is a 30.5 Hz signal at 16 MHz. Refer to Listing 6. By changing the PCA clock input, slower waveforms can also be generated.

Listing 6. High Speed Output (Without Interrupt)

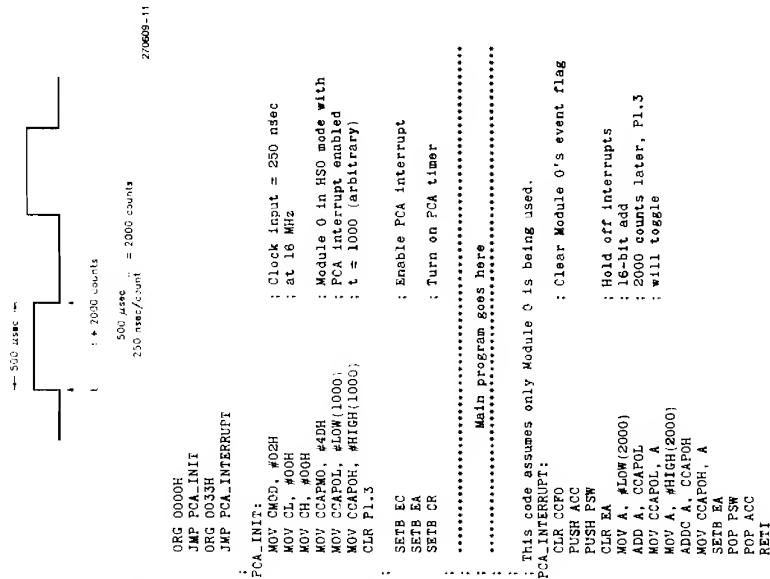
```

: Maximum output with HSO mode without interrupts = 30.5 Hz signal
: Frequency = 16 MHz
: PCA clock input = 1/4 x Fosc -> 250 nsec
:
MOV CMOD, #02H
MOV CL, #00H
MOV CH, #00H
MOV CCAPM0, #4CH : HSO mode without interrupt enabled
MOV CCAPOL, #0FFH : Write to low byte first
MOV CCAPOH, #0FFH : P1.3 will toggle every 216 counts
: or 16.4 msec
: Period = 30.5 Hz
: Turn on PCA timer
SETB CR

```

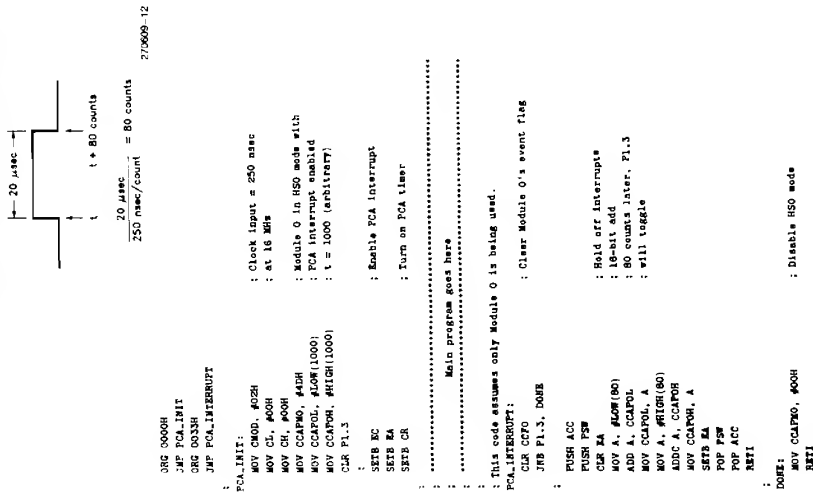
In this next example, the PCA interrupt is used to change the compare value for each toggle. This way a variable frequency output can be generated. Listing 7 shows an output of 1 KHz at 16 Mhz.

Listing 7: High Speed Output (With Interrupt)



Another option with the HSO mode is to generate a single pulse. Listing 8 shows the code for an output with a pulse width of 20 μsec. As in the previous example, the PCA interrupt will be used to change the time for the toggle. The first toggle will occur at time 't'. After 80 counts of the PCA timer, 20 μsec will have expired, and the next toggle will occur. Then the HSO mode will be disabled.

Listing 8: High Speed Output (Single Pulse)



WATCHDOG TIMER

An on-board watchdog timer is available with the PCA to improve the reliability of the system without increasing chip count. Watchdog timers are useful for systems which are susceptible to noise, power glitches, or electrostatic discharge. Module 4 is the only PCA module which can be programmed as a watchdog. However, this module can still be used for other modes if the watchdog is not needed.

Figure 11 shows a diagram of how the watchdog works. The user pre-loads a 16-bit value in the compare registers. Just like the other compare modes, this 16-bit value is compared to the PCA timer value. If a match is allowed to occur, an internal reset will be generated. This will not cause the RST pin to be driven high.

In order to hold off the reset, the user has three options: (1) periodically change the compare value so it will never match the PCA timer, (2) periodically change the PCA timer value so it will never match the compare value, or (3) disable the watchdog by clearing the WDTE bit before a match occurs and then re-enable it.

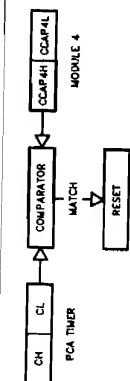


Figure 11. Watchdog Timer Mode (Module 4)

270600-19

Listing 9. Watchdog Timer.

```
INIT_WATCHDOG:
MOV CCAP4M, #4CH
MOV CCAP4L, #0FFH
MOV CCAP4H, #0FFH
;
; Module 4 in compare mode
; Write to low byte first
; Before PCA timer counts up to
; 0FFF Hex, these compare values
; must be changed
; Set the WDTE bit to enable the
; watchdog timer without changing
; the other bits in CMOD
;
; Main program goes here, but CALL WATCHDOG periodically.
;
WATCHDOG:
CLR EA
MOV CCAP4L, #00H
MOV CCAP4H, CH
SETB EA
RET
```

PULSE WIDTH MODULATOR

The PCA can generate 8-bit PWMs by comparing the low byte of the PCA timer (CL) with the low byte of the compare registers (CCAPnL). When $CL < CCAPnL$, the output is low. When $CL \geq CCAPnL$, the output is high.

To control the duty cycle of the output, the user actually loads a value into the high byte CCAPnH (see Figure 12). Since a write to this register is asynchronous, a new value is not shifted into CCAPnL for comparison until

the next period of the output: that is, when CL rolls over from 255 to 00. This mechanism provides "glitch-free" writes to CCAPnH when the duty cycle of the output is changed.

CCAPnH can contain any integer from 0 to 255, but Figure 13 shows a few common duty cycles and the corresponding values for CCAPnH. Note that a 0% duty cycle can be obtained by writing to the port pin directly with the CLR bit instruction. To calculate the CCAPnH value for a given duty cycle, use the following equation:

$$CCAPnH = 256 (1 - \text{Duty Cycle})$$

where CCAPnH is an 8-bit integer and Duty Cycle is expressed as a fraction.

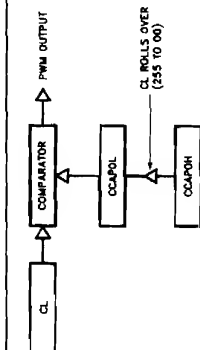


Figure 12. PWM Mode (Module 0)

270600-14

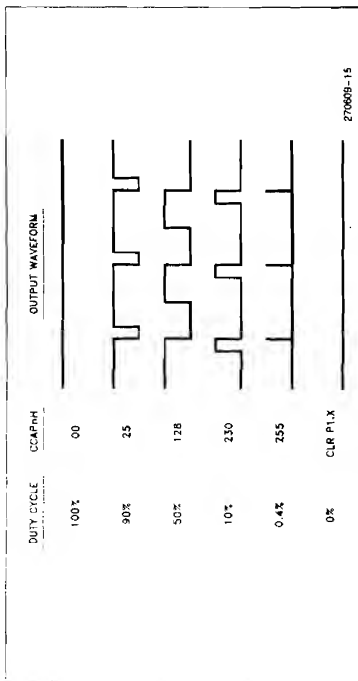


Figure 13. CCAPnH Varies Duty Cycle

Table 4. PWM Frequencies.

PCA Timer Mode	PWM Frequency	
	12 MHz	16 MHz
1/12 Osc. Frequency	3.9 KHz	5.2 KHz
1/4 Osc. Frequency	11.8 KHz	15.6 KHz
Timer 0 Overflow:		
8-bit	15.5 Hz	20.3 Hz
16-bit	0.08 Hz	0.08 Hz
8-bit Auto-Reload	3.9 KHz to 15.3 Hz	5.2 KHz to 20.3 Hz
External Input (Max)	5.9 KHz	7.8 KHz

Listing 10. PWM

```

INIT-PWM:
MOV CMOD, #02H
MOV CL, #00H
MOV CH, #00H
MOV CCAPMO, #42H
MOV CCAPOL, #00H
MOV CCAPDH, #128D
;
; 50 percent duty cycle
; Turn on PCA timer
SETB CR

```

CONCLUSION

This list of examples with the PCA is by no means exhaustive. However, the advantages of the PCA can easily be seen from the given applications. For example, the PCA can provide better resolution than Timers 0, 1 and 2 because the PCA clock rate can be three times faster. The PCA can also perform many tasks that these hardware timers can not, i.e. measure phase differences between signals or generate PWMs. In a sense, the PCA provides the user with five more timer/counters in addition to Timers 0, 1 and 2 on the 8XC31FA/FB.

Appendix A includes test routines for all the software examples in this application note. The divide routine for calculating duty cycles is in Appendix B. Additionally, Appendix C is a table of the Special Function Registers for the 8XC31FA/FB with the new or modified registers **boldfaced**.

The frequency of the PWM output will depend on which of the four inputs is chosen for the PCA timer. The maximum frequency is 15.6 KHz at 16 MHz. Refer to Table 4 for a summary of the different PWM frequencies possible with the PCA.

Listing 10 shows how to initialize Module 0 for a PWM signal at 50% duty cycle. Notice that no PCA interrupt is needed to generate the PWM (i.e. no software overhead). To create a PWM output on the 8031 requires a hardware timer plus software overhead to toggle the port pin. The advantage of the PCA is obvious, not to mention it can support up to 5 PWM outputs with just one chip.

APPENDIX A TEST ROUTINES

```

;-----
; Module 0 - Measuring Pulse Width
;-----
; Parameters
;-----
; Pin: PC4
; Prescaler: 64
; Slew Rate: Disabled
; Signal: Square Wave
;-----
; Variables
;-----
DATA SEGMENT
    PULSE_WIDTH DB ?
    FLAG DB ?
BIT EQU 0
;-----
CAPTURE PROC
    ORG 0000H
    JMP PCA_INT
ORG 0003H
    JMP PCA_INTERRUPT
PCA_INT:
    INVDATA PCA_TIMER
    MOV CH, 000H
    MOV CL, 00
    MOV CL, 00
    ; Initialize Module 0 in capture mode
    MOV CCAPM0, 0111H
    MOV CCAPR0, 000
    MOV CCAPL0, 000
    SETB EC
    SETB EA
    SETB CR
    CLR FLAG
    ; Enable PCA Interrupt
    ; Turn PCA timer on
    ; Clear test flag
    ; Wait for PCA interrupt
    WAIT:
        JMP $
        JMP WAIT
;-----
; Test program only
;-----
TEST PROC
    ; Module 0 is the only module being used, if
    ; other PCA module's are being tested, software must check which
    ; module's event flag caused the interrupt.
    PCA_INTERRUPT:
        CLR CDF0
        JB FLAG, SECOND_CAPTURE
        FIRST_CAPTURE:
            MOV CAPTURE, CCAPR0
            MOV CAPTURE+1, CCAPL0
            ; Clear module 0's event flag
;-----

```

270809-18

- Change module to new capture
- falling edges
- Slightly first capture complete

```

MOV CCAPM0, #11H
SETB FLAG
RETI

SECOND_CAPTURE:
PUSH ACC
PUSH PSW
CLR C
MOV A, CCAP0L
SUBB A, CAPTURE
MOV PULSE_WIDTH, A
MOV CAPTURE+1, C
SUBB A, CAPTURE+1
MOV PULSE_WIDTH+1, A

MOV CCAPM0, #11H
CLR FLAG
PUSH PSW
POP ACC
RETI

```

END

270609-7

```

; Signify first capture complete
SETI FLAG
RETI

SECOND_CAPTURE:
PUSH ACC
PUSH PSW
CLR C
MOV A, C000H
MOV B, C000H
MOV C, C000H
MOV PERIOD, A
MOV A, C000H
SUBH A, C000H-1
MOV PERIOD-1, A

CLR FLAG
POP PSW
POP ACC
RETI

```

END

Listing 1b - Measuring Periods

```
$nomod51
$nosymbols
$nofist
$include (req)
```

Verluste

CAPTURE PERIOD	DATA	30H
PERIOD	DATA	32H
FLAG	BIT	30H 0

PAGE 0000H

NEW YORK

JMP PCA_INTERRUPT

Initiated PCA timer

PCA_INIT: MOV CMOD, #

MOV CH, 800H

MOV CL, 800

```
5 Initialize Module 0 In
```

MOV CCAPM0

MOV CCAP,0FH

10

038136
SETB EC

SETB EA
SETB C0SEIBER
CLIF FLAG

LESS FADING

Test program only

WAST.

JMP \$
JMP WAIT

This code assumes only Module 0 is being used. If other modules are being used, software must check which module's flag caused the failure.

PC-A INTERPRET.

CLA CCE0

JB FLAG, SEC

11

FIRST_CAPTURE:

MOV CAPTURE

MOV CAPTURE

270609-18

LISTING 2. MASS-100A PARAMETERS

```
$nomodel;
$noasmm;
$include 'mas102.tbl'
$part
```

Variables

```
CAPTURE      20H
PERIOD       32H
SAMPLE_COUNT 34H
FLAG         20H.0
BIT          20H.0
```

```
ORG 0000H
JMP PCA_INIT
```

```
ORG 0030H
JMP PCA_INTERRUPT
```

```
INITIALIZE PCA timer
PCA_INIT:  MOV CMOOD, #00H
          MOV COUNT, #0
          MOV CL, #0
```

```
; Initialize Module 0 in capture mode
```

```
MOV CCAPMOD, #23H
```

```
MOV CCAPDH, #00
```

```
MOV CCAPDL, #00
```

```
MOV SAMPLE_COUNT, #100
```

```
SETB EC
```

```
SETB EA
```

```
SETB CLR
```

```
CLR FLAG
```

```
Test program only
```

```
WAIT:  JMP $
```

```
      JMP WAIT
```

```
      ; Wait for PCA interrupt
```

```
      ; Wait for PCA interrupt
```

```
      ; Wait for PCA interrupt
```

```
      ; Wait for PCA interrupt
```

```
      ; Wait for PCA interrupt
```

```
      ; Wait for PCA interrupt
```

```
      ; Wait for PCA interrupt
```

```
      ; Wait for PCA interrupt
```

```
      ; Wait for PCA interrupt
```

```
      ; Wait for PCA interrupt
```

```
      ; Wait for PCA interrupt
```

```
      ; Wait for PCA interrupt
```

```
      ; Wait for PCA interrupt
```

```
      ; Wait for PCA interrupt
```

```
      ; Wait for PCA interrupt
```

```
      ; Wait for PCA interrupt
```

```
      ; Wait for PCA interrupt
```

```
      ; Wait for PCA interrupt
```

```
      ; Wait for PCA interrupt
```

```
      ; Wait for PCA interrupt
```

```
      ; Wait for PCA interrupt
```

```
      ; Wait for PCA interrupt
```

```
      ; Wait for PCA interrupt
```

```
      ; Wait for PCA interrupt
```

```
      ; Wait for PCA interrupt
```

```
      ; Wait for PCA interrupt
```

```
      ; Wait for PCA interrupt
```

```
      ; Wait for PCA interrupt
```

```
      ; Wait for PCA interrupt
```

```
      ; Wait for PCA interrupt
```

```
      ; Wait for PCA interrupt
```

```
      ; Wait for PCA interrupt
```

```
      ; Wait for PCA interrupt
```

```
FIRST_CAPTURE:
MOV CAPTURE, CCAPDL
MOV CAPTURE+1, CCAPDH
SETB FLAG
RETI

; Signify first capture complete

NEXT_CAPTURE:
DJNZ SAMPLE_COUNT, EXIT
PUSH ACC
PUSH PSW
RETI
; 15-Bit subtraction
MOV A, CCAPDL
SUBB A, CAPTURE
MOV PERIOD, A
MOV A, CCAPDH
SUBB A, CAPTURE+1
MOV PERIOD+1, A
MOV SAMPLE_COUNT, #100
CAPTURE
CAPTURE+1
POP PSW
POP ACC
RETI

; Reload for next capture

EXIT:
END
```

270609-21

270609-20

Listing 3. Measurement Data

```

;-----
; Symbols
;-----
%include "ap252.pdf"
;-----
; Variables
;-----
DATA          20H
PULSE_WIDTH  32H
PERIOD       34H
BIT          20H:0
BIT          20H:1
FLAG_1      20H:0
FLAG_2      20H:1
ORG 0000H
JMP PCA_INT
ORG 0030H
JMP PCA_INTERRUPT
;-----
; Initialize PCA User
PCA_INT:      MOV CH, 000H
              MOV CL, 000
              MOV C, 000
              MOV CAPH, 000
              MOV CAPL, 000
              MOV CAPM, 021H
              MOV CAPPH, 000
              MOV CAPPL, 000
              CLR FLAG_1
              CLR FLAG_2
              SETB EC
              SETB EA
              SETB CR
;-----
; Test program only
;-----
WAIT:         JMP $
              JMP WAIT
;-----
; Wait for PCA Interrupt
;-----
PCA_INTERRUPT: CLR CCRB
              JIB FLAG_1, SECOND_CAPTURE

```

This code assumes Module 0 is the only PCA module being used.

270606-22

```

FIRST_CAPTURE:
MOV CAPTURE, CCAPIRL
MOV CAPTURE+1, CCAPIRH
MOV PERIOD, 000H
MOV CCAPIRL, 021H
RET

SECOND_CAPTURE:
PUSH ACC
PUSH PSW
JIB FLAG_2, THIRD_CAPTURE
CLR C
MOV A, CCAPIRL
SUBB A, CAPTURE
MOV PULSE_WIDTH, A
MOV A, CCAPIRH
SUBB A, CAPTURE+1
MOV PULSE_WIDTH+1, A
SETB FLAG_2
POP PSW
POP ACC
RET

THIRD_CAPTURE:
CLR C
MOV A, CCAPIRL
SUBB A, CAPTURE
MOV PERIOD, A
MOV A, CCAPIRH
SUBB A, CAPTURE+1
MOV PERIOD+1, A
MOV CCAPIRL, 021H
CLR FLAG_1
CLR FLAG_2
POP PSW
POP ACC
RET

```

END

270606-23

270609-24

JB FLAG 0. MOD0 LEADING

JB FLAG_1, MOD1_LEADING

MOD0_LEADING:

; 16-bit subtraction

```

SUBB A, CAPTURE_1
MOV PHASE_A
MOV A, CAPTURE_1+1
MOV PHASE_CAPTURE_0+1
MOV PHASE+1, A
CLR FLAG_0
JMP EXIT

```

MOD1_LEADING:

; 16-bit subtraction

```

SUBB A, CAPTURE_0
SUBB A, CAPTURE_1
MOV PHASE_A
MOV A, CAPTURE_0+1
SUBB A, CAPTURE_1+1
MOV PHASE+1, A
CLR FLAG_1

```

```

EXIT:
POP PSW
POP ACC
RETI

```

END

270606-26

Listing 5. Software Timer

```

$noerrors
$asmmacos
$notatt
$include (reg252.pdf)
$list
; Software Timer mode which interrupts every 20 msec with Fosc = 12 MHz

ORG 0000H
JMP PCA_INIT
ORG 0003H
JMP PCA_INTERRUPT

; Initialize PCA timer
PCA_INIT
MOV CMO0, #00H
MOV CMO1, #00H
MOV CL, #00H

MOV CCAPM0, #00H
MOV CCAPM1, #00H
MOV CCAPM0, #HIGH(20000)
MOV CCAPM1, #HIGH(20000)

SETB EC
SETB EA
SETB CR

; Input to PCA timer = 1/12 * Fosc
; Software Timer mode with interrupt
; Write to low byte first
; Enable PCA interrupt
; Turn PCA timer on

; Test program only
WAIT:
JMP $
JMP WAIT

; Wait for PCA interrupt

This code assumes Module 0 is the only module being used. If
other PCA module's are being used, software must check which
module's event flag caused the interrupt.

PCA_INTERRUPT:
CLR CCF0
PUSH ACC
PUSH PSW
CLR EA
MOV A, #LOW(20000)
ADD A, CCAPM0
MOV CCAPM0, A
MOV A, #HIGH(20000)
MOV A, CCAPM1
MOV CCAPM1, A
SETB EA
POP PSW
POP ACC
RETI

```

END

270606-27

Listing 7. High Speed Outout (with Interrupts)

```
$nomod$1
$nosymbols
$noflist
$include (reg
alist
```

variable frequency. This example outputs a 1KHz signal.

ORIG 00000H
IMP PCA INIT

```
ORG 003H  
JMP PCA_INTERRUPT
```

```

; HSO mode with interrupt enabled
; t = 1000 arbitrary
CAPM0, #40H
CAPOL, #LOW(1000)
CAPOH, #HIGH(1000)

```

Figure 1

270609-28

WALSH

UPWARD

This code assumes Module 0 is the only module being used. If other PCA module's are being used, software must check which module's event flag caused the interrupt.

PCA INTERRUPT:

```
F0      ; Clear module 0's event flag
CC      ;
SW      ;
        ; Hold off interrupts
        ; 16-bit add
        ; 2000 counts later P1.3
        ; will toggle
```

ON

27089-30

Listing 2. Watchdog Timer

[illegible]

270609-33

270809-32

END

270608-34

DUTY_CYCLE_CALCULATION

CALCULATES DUTY CYCLE = PULSE WIDTH / PERIOD

Inputs to this routine are 16-bit pulse width and period measurements of a rectangular waveform. The output is a 9-bit BCD number representing the duty cycle of the waveform. The low 8 bits of the result are returned in DUTY_CYCLE. The 9th bit is the carry bit in the PSW. If the duty cycle is between 0 and 99 percent, the carry bit is 0 and DUTY_CYCLE contains the two BCD digits representing the duty cycle as a percent. If the duty cycle is 100 percent, the carry bit is 1 and DUTY_CYCLE contains 0.

INPUTS: PULSE_WIDTH 2 bytes in externally defined DATA
 (low: byte of PULSE_WIDTH, high byte of PULSE_WIDTH+1)

PERIOD 2 bytes in externally defined DATA flow byte at PERIOD high byte at PERIOD+1)

QUITBIT: DUTY CYCLE 1 byte in externally defined DATA

VARIABLES AND REGISTERS MODIFIED:

PULSE WIDTH, DUTY CYCLE

ACC, B, PSW, R2, R3

ERROR EXT: Err with QV = 1 indicating PULSE WIDTH > PERIOD.

DUTY CYCLE CALCULATION:

```
MOV A,PERIOD+1
CJNE A,PULSE_WIDTH+1,NOT_EQUAL
MOV A,PERIOD
CJNE A,PULSE_WIDTH,NOT_EQUAL
```

```

EQUAL: SETB C
      MOV DUTY_CYCLE,#0
      CLR OV
      RET

NOT_EQUAL: CONTINUE
      JNC SETB_OV
      SETB OV
      RET

CONTINUE:
      MOV R4,#CYCLE,#0
      MOV R4,#A
      RET

TIMES_TWO:
      MOV R4,#A_PULSE_WIDTH
      RLC
      MOV PULSE_WIDTH,A
      MOV A,PULSE_WIDTH+1
      RLC
      MOV A,PULSE_WIDTH+1,A
      MOV A,R3
      RLC
      MOV R4,A
      COMPARE:
      CINE R4,#FINAL_DONE
      MOV A,PULSE_WIDTH+1
      CINE SUBB,#FINAL_DONE
      MOV A,PULSE_WIDTH
      CINE A,PERIOD,FINAL_DONE
      JNC DONE
      CINE R4,#A
      MOV R4,A
      BUILD_DUTY_CYCLE:
      MOV A,DUTY_CYCLE
      RLC
      MOV A,DUTY_CYCLE
      MOV ACC,LOOP_CONTROL
      SUBTRACT:
      MOV A,PULSE_WIDTH
      SUBB A,PERIOD
      MOV PULSE_WIDTH,A
      MOV A,PULSE_WIDTH+1
      SUBB A,PERIOD+1
      MOV PULSE_WIDTH+1,A
      MOV A,R3
      SUBB A,R4
      MOV R4,A
      LOOP_CONTROL:
      DJNZ R2,TIMES_TWO

```

27/0609-36

```

FINAL_TIMES_TWO:
      MOV A,PULSE_WIDTH
      RLC
      MOV PULSE_WIDTH,A
      MOV A,PULSE_WIDTH+1
      RLC
      MOV PULSE_WIDTH+1,A
      MOV A,R3
      RLC
      MOV R4,A
      FINAL_COMPARE:
      CINE R4,#FINAL_DONE
      MOV A,PULSE_WIDTH+1
      CINE SUBB,#FINAL_DONE
      MOV A,PULSE_WIDTH
      CINE A,PERIOD,FINAL_DONE
      JNC DONE
      CINE R4,#A
      MOV R4,A
      CONVERT_TO_BCD:
      MOV A,DUTY_CYCLE
      ADD A,#1
      MOV DUTY_CYCLE,A
      JNC CONVERT_TO_BCD
      CLR OV
      RET

CONVERT_TO_BCD:
      MOV A,DUTY_CYCLE
      MOV B,#10
      MUL AB
      XCH AB
      SWAP A
      MOV DUTY_CYCLE,A
      MOV A,#10
      MUL AB
      XCH AB
      MOV DUTY_CYCLE,A
      MOV A,#10
      MUL AB
      MOV A,B
      CINE A,#10
      MOV A,B
      TEST:
      CINE A,#10
      MOV A,DUTY_CYCLE
      ADD A,#1
      DA A
      MOV DUTY_CYCLE,A
      OUT: RET
      END

```

27/0609-37

APPENDIX C

A map of the Special Function Register (SFR) space is shown in Table A1. Those registers which are new or have new bits added for the 80C51FA and 80C51FB have been **boldfaced**.

Note that not all of the addresses are occupied. Unoccupied addresses are not implemented on the chip.

Read accesses to these addresses will in general return random data, and write accesses will have no effect.

User software should not write to these unimplemented locations, since they may be used in future 80C51 family products to invoke new features. In that case the reset or inactive values of the new bits will always be 0, and their active values will be 1.

Table A1. Special Function Register Memory Map and Values After Reset

FF	CH	CCAP0H	CCAP1H	CCAP2H	CCAP3H	CCAP4H	FF
F0	* B	00000000	XXXXXXXX	XXXXXXXX	XXXXXXXX	XXXXXXXX	F7
E8	CL	00000000	XXXXXXXX	CCAP2L	CCAP3L	CCAP4L	EF
E0	* ACC	00000000	XXXXXXXX	XXXXXXXX	XXXXXXXX	XXXXXXXX	E7
D8	CCON	00000000	CCAPM0	CCAPM1	CCAPM2	CCAPM3	DF
D0	* PSW	00000000	XXXXXXXX	XXXXXXXX	XXXXXXXX	XXXXXXXX	D7
C8	T2CON	00000000	T2MOD	TL2	TH2	00000000	CF
C0							C7
B8	* IP	SADEN					BF
B0	* P3	11111111					B7
A8	* IE	SADDR					AF
A0	* P2	11111111					A7
98	* SCON	SBUF					9F
90	* P1	11111111					97
88	* TCON	TMOD	* TL0	* TH0	* TH1		8F
80	* P0	* SP	* DPL	* DPH		* PCON	* 87
						00XX0000	

* = Bit in the 8051 core (See 9051 Hardware Description in the Embedded Controller Handbook for explanations of these bits).
 ** = Same description as 80C51F0, 80C51F1, 80C51F2, 80C51F3, 80C51F4, 80C51F5, 80C51F6, 80C51F7, 80C51F8, 80C51F9, 80C51FA, 80C51FB, 80C51FC, 80C51FD, 80C51FE, 80C51FF, 80C51G0, 80C51G1, 80C51G2, 80C51G3, 80C51G4, 80C51G5, 80C51G6, 80C51G7, 80C51G8, 80C51G9, 80C51GA, 80C51GB, 80C51GC, 80C51GD, 80C51GE, 80C51GF, 80C51GG, 80C51GH, 80C51GI, 80C51GJ, 80C51GK, 80C51GL, 80C51GM, 80C51GN, 80C51GO, 80C51GP, 80C51GQ, 80C51GR, 80C51GS, 80C51GT, 80C51GU, 80C51GV, 80C51GW, 80C51GX, 80C51GY, 80C51GZ, 80C51H0, 80C51H1, 80C51H2, 80C51H3, 80C51H4, 80C51H5, 80C51H6, 80C51H7, 80C51H8, 80C51H9, 80C51HA, 80C51HB, 80C51HC, 80C51HD, 80C51HE, 80C51HF, 80C51HG, 80C51HH, 80C51HI, 80C51HJ, 80C51HK, 80C51HL, 80C51HM, 80C51HN, 80C51HO, 80C51HP, 80C51HQ, 80C51HR, 80C51HS, 80C51HT, 80C51HU, 80C51HV, 80C51HW, 80C51HX, 80C51HY, 80C51HZ, 80C51I0, 80C51I1, 80C51I2, 80C51I3, 80C51I4, 80C51I5, 80C51I6, 80C51I7, 80C51I8, 80C51I9, 80C51IA, 80C51IB, 80C51IC, 80C51ID, 80C51IE, 80C51IF, 80C51IG, 80C51IH, 80C51II, 80C51IJ, 80C51IK, 80C51IL, 80C51IM, 80C51IN, 80C51IO, 80C51IP, 80C51IQ, 80C51IR, 80C51IS, 80C51IT, 80C51IU, 80C51IV, 80C51IW, 80C51IX, 80C51IY, 80C51IZ, 80C51J0, 80C51J1, 80C51J2, 80C51J3, 80C51J4, 80C51J5, 80C51J6, 80C51J7, 80C51J8, 80C51J9, 80C51JA, 80C51JB, 80C51JC, 80C51JD, 80C51JE, 80C51JF, 80C51JG, 80C51JH, 80C51JI, 80C51JJ, 80C51JK, 80C51JL, 80C51JM, 80C51JN, 80C51JO, 80C51JP, 80C51JQ, 80C51JR, 80C51JS, 80C51JT, 80C51JU, 80C51JV, 80C51JW, 80C51JX, 80C51JY, 80C51JZ, 80C51K0, 80C51K1, 80C51K2, 80C51K3, 80C51K4, 80C51K5, 80C51K6, 80C51K7, 80C51K8, 80C51K9, 80C51KA, 80C51KB, 80C51KC, 80C51KD, 80C51KE, 80C51KF, 80C51KG, 80C51KH, 80C51KI, 80C51KJ, 80C51KK, 80C51KL, 80C51KM, 80C51KN, 80C51KO, 80C51KP, 80C51KQ, 80C51KR, 80C51KS, 80C51KT, 80C51KU, 80C51KV, 80C51KW, 80C51KX, 80C51KY, 80C51KZ, 80C51L0, 80C51L1, 80C51L2, 80C51L3, 80C51L4, 80C51L5, 80C51L6, 80C51L7, 80C51L8, 80C51L9, 80C51LA, 80C51LB, 80C51LC, 80C51LD, 80C51LE, 80C51LF, 80C51LG, 80C51LH, 80C51LI, 80C51LJ, 80C51LK, 80C51LL, 80C51LM, 80C51LN, 80C51LO, 80C51LP, 80C51LQ, 80C51LR, 80C51LS, 80C51LT, 80C51LU, 80C51LV, 80C51LW, 80C51LX, 80C51LY, 80C51LZ, 80C51M0, 80C51M1, 80C51M2, 80C51M3, 80C51M4, 80C51M5, 80C51M6, 80C51M7, 80C51M8, 80C51M9, 80C51MA, 80C51MB, 80C51MC, 80C51MD, 80C51ME, 80C51MF, 80C51MG, 80C51MH, 80C51MI, 80C51MJ, 80C51MK, 80C51ML, 80C51MM, 80C51MN, 80C51MO, 80C51MP, 80C51MQ, 80C51MR, 80C51MS, 80C51MT, 80C51MU, 80C51MV, 80C51MW, 80C51MX, 80C51MY, 80C51MZ, 80C51N0, 80C51N1, 80C51N2, 80C51N3, 80C51N4, 80C51N5, 80C51N6, 80C51N7, 80C51N8, 80C51N9, 80C51NA, 80C51NB, 80C51NC, 80C51ND, 80C51NE, 80C51NF, 80C51NG, 80C51NH, 80C51NI, 80C51NJ, 80C51NK, 80C51NL, 80C51NM, 80C51NN, 80C51NO, 80C51NP, 80C51NQ, 80C51NR, 80C51NS, 80C51NT, 80C51NU, 80C51NV, 80C51NW, 80C51NX, 80C51NY, 80C51NZ, 80C51O0, 80C51O1, 80C51O2, 80C51O3, 80C51O4, 80C51O5, 80C51O6, 80C51O7, 80C51O8, 80C51O9, 80C51OA, 80C51OB, 80C51OC, 80C51OD, 80C51OE, 80C51OF, 80C51OG, 80C51OH, 80C51OI, 80C51OJ, 80C51OK, 80C51OL, 80C51OM, 80C51ON, 80C51OO, 80C51OP, 80C51OQ, 80C51OR, 80C51OS, 80C51OT, 80C51OU, 80C51OV, 80C51OW, 80C51OX, 80C51OY, 80C51OZ, 80C51P0, 80C51P1, 80C51P2, 80C51P3, 80C51P4, 80C51P5, 80C51P6, 80C51P7, 80C51P8, 80C51P9, 80C51PA, 80C51PB, 80C51PC, 80C51PD, 80C51PE, 80C51PF, 80C51PG, 80C51PH, 80C51PI, 80C51PJ, 80C51PK, 80C51PL, 80C51PM, 80C51PN, 80C51PO, 80C51PP, 80C51PQ, 80C51PR, 80C51PS, 80C51PT, 80C51PU, 80C51PV, 80C51PW, 80C51PX, 80C51PY, 80C51PZ, 80C51Q0, 80C51Q1, 80C51Q2, 80C51Q3, 80C51Q4, 80C51Q5, 80C51Q6, 80C51Q7, 80C51Q8, 80C51Q9, 80C51QA, 80C51QB, 80C51QC, 80C51QD, 80C51QE, 80C51QF, 80C51QG, 80C51QH, 80C51QI, 80C51QJ, 80C51QK, 80C51QL, 80C51QM, 80C51QN, 80C51QO, 80C51QP, 80C51QQ, 80C51QR, 80C51QS, 80C51QT, 80C51QU, 80C51QV, 80C51QW, 80C51QX, 80C51QY, 80C51QZ, 80C51R0, 80C51R1, 80C51R2, 80C51R3, 80C51R4, 80C51R5, 80C51R6, 80C51R7, 80C51R8, 80C51R9, 80C51RA, 80C51RB, 80C51RC, 80C51RD, 80C51RE, 80C51RF, 80C51RG, 80C51RH, 80C51RI, 80C51RJ, 80C51RK, 80C51RL, 80C51RM, 80C51RN, 80C51RO, 80C51RP, 80C51RQ, 80C51RR, 80C51RS, 80C51RT, 80C51RU, 80C51RV, 80C51RW, 80C51RX, 80C51RY, 80C51RZ, 80C51S0, 80C51S1, 80C51S2, 80C51S3, 80C51S4, 80C51S5, 80C51S6, 80C51S7, 80C51S8, 80C51S9, 80C51SA, 80C51SB, 80C51SC, 80C51SD, 80C51SE, 80C51SF, 80C51SG, 80C51SH, 80C51SI, 80C51SJ, 80C51SK, 80C51SL, 80C51SM, 80C51SN, 80C51SO, 80C51SP, 80C51SQ, 80C51SR, 80C51SS, 80C51ST, 80C51SU, 80C51SV, 80C51SW, 80C51SX, 80C51SY, 80C51SZ, 80C51T0, 80C51T1, 80C51T2, 80C51T3, 80C51T4, 80C51T5, 80C51T6, 80C51T7, 80C51T8, 80C51T9, 80C51TA, 80C51TB, 80C51TC, 80C51TD, 80C51TE, 80C51TF, 80C51TG, 80C51TH, 80C51TI, 80C51TJ, 80C51TK, 80C51TL, 80C51TM, 80C51TN, 80C51TO, 80C51TP, 80C51TQ, 80C51TR, 80C51TS, 80C51TT, 80C51TU, 80C51TV, 80C51TW, 80C51TX, 80C51TY, 80C51TZ, 80C51U0, 80C51U1, 80C51U2, 80C51U3, 80C51U4, 80C51U5, 80C51U6, 80C51U7, 80C51U8, 80C51U9, 80C51UA, 80C51UB, 80C51UC, 80C51UD, 80C51UE, 80C51UF, 80C51UG, 80C51UH, 80C51UI, 80C51UJ, 80C51UK, 80C51UL, 80C51UM, 80C51UN, 80C51UO, 80C51UP, 80C51UQ, 80C51UR, 80C51US, 80C51UT, 80C51UU, 80C51UV, 80C51UW, 80C51UX, 80C51UY, 80C51UZ, 80C51V0, 80C51V1, 80C51V2, 80C51V3, 80C51V4, 80C51V5, 80C51V6, 80C51V7, 80C51V8, 80C51V9, 80C51VA, 80C51VB, 80C51VC, 80C51VD, 80C51VE, 80C51VF, 80C51VG, 80C51VH, 80C51VI, 80C51VJ, 80C51VK, 80C51VL, 80C51VM, 80C51VN, 80C51VO, 80C51VP, 80C51VQ, 80C51VR, 80C51VS, 80C51VT, 80C51VU, 80C51VV, 80C51VW, 80C51VX, 80C51VY, 80C51VZ, 80C51W0, 80C51W1, 80C51W2, 80C51W3, 80C51W4, 80C51W5, 80C51W6, 80C51W7, 80C51W8, 80C51W9, 80C51WA, 80C51WB, 80C51WC, 80C51WD, 80C51WE, 80C51WF, 80C51WG, 80C51WH, 80C51WI, 80C51WJ, 80C51WK, 80C51WL, 80C51WM, 80C51WN, 80C51WO, 80C51WP, 80C51WQ, 80C51WR, 80C51WS, 80C51WT, 80C51WU, 80C51WV, 80C51WW, 80C51WX, 80C51WY, 80C51WZ, 80C51X0, 80C51X1, 80C51X2, 80C51X3, 80C51X4, 80C51X5, 80C51X6, 80C51X7, 80C51X8, 80C51X9, 80C51XA, 80C51XB, 80C51XC, 80C51XD, 80C51XE, 80C51XF, 80C51XG, 80C51XH, 80C51XI, 80C51XJ, 80C51XK, 80C51XL, 80C51XM, 80C51XN, 80C51XO, 80C51XP, 80C51XQ, 80C51XR, 80C51XS, 80C51XT, 80C51XU, 80C51XV, 80C51XW, 80C51XX, 80C51XY, 80C51XZ, 80C51Y0, 80C51Y1, 80C51Y2, 80C51Y3, 80C51Y4, 80C51Y5, 80C51Y6, 80C51Y7, 80C51Y8, 80C51Y9, 80C51YA, 80C51YB, 80C51YC, 80C51YD, 80C51YE, 80C51YF, 80C51YG, 80C51YH, 80C51YI, 80C51YJ, 80C51YK, 80C51YL, 80C51YM, 80C51YN, 80C51YO, 80C51YP, 80C51YQ, 80C51YR, 80C51YS, 80C51YT, 80C51YU, 80C51YV, 80C51YW, 80C51YX, 80C51YY, 80C51YZ, 80C51Z0, 80C51Z1, 80C51Z2, 80C51Z3, 80C51Z4, 80C51Z5, 80C51Z6, 80C51Z7, 80C51Z8, 80C51Z9, 80C51ZA, 80C51ZB, 80C51ZC, 80C51ZD, 80C51ZE, 80C51ZF, 80C51ZG, 80C51ZH, 80C51ZI, 80C51ZJ, 80C51ZK, 80C51ZL, 80C51ZM, 80C51ZN, 80C51ZO, 80C51ZP, 80C51ZQ, 80C51ZR, 80C51ZS, 80C51ZT, 80C51ZU, 80C51ZV, 80C51ZW, 80C51ZX, 80C51ZY, 80C51ZZ

Small DC Motor Control